UNIT-1

Introduction to NLP

Syllabus

Introduction to NLP: Origins and Challenges, Language and Grammar in NLP, Regular Expressions and Finite-State Automata, Tokenization: Text Segmentation and Sentence Splitting, Morphological Parsing: Stemming and Lemmatization, Spelling Error Detection and Correction, Minimum Edit Distance and Applications, Statistical Language Models: Unigram, Bigram, and Trigram Models, Processing Indian Languages in NLP

Introduction to NLP:

- **Natural Language Processing (NLP)** is a branch of **Artificial Intelligence (AI)** that deals with the interaction between computers and human language.
- It enables machines to **understand, interpret, and generate** natural language in the form of text or speech.
- NLP combines techniques from **computer science, linguistics, and machine learning** to process language.
- It involves tasks such as **tokenization, part-of-speech tagging, parsing, sentiment analysis, and machine translation**.
- NLP is widely used in applications like **chatbots, search engines, spell checkers, voice assistants, and language translators**.
- By using NLP, computers can communicate with humans in a more natural and meaningful way.

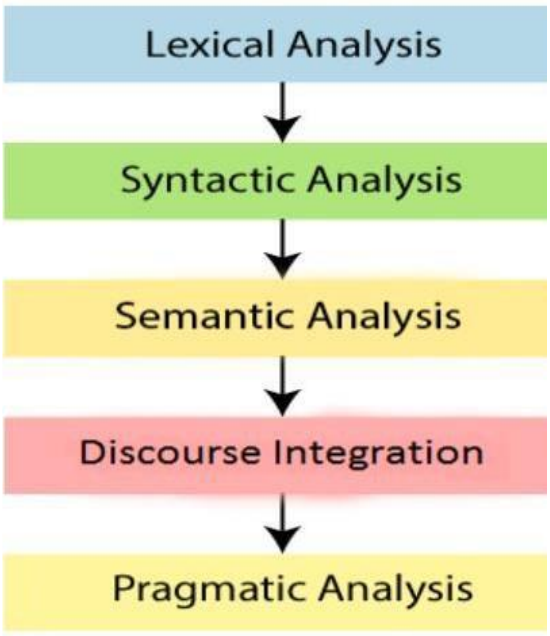
Why it is important:

Human language is complex, ambiguous, and context-dependent. NLP bridges the gap between human communication and machine understanding, making interaction with computers more natural.

Examples:

- Asking questions to Google Assistant
- Chatting with chatbots
- Automatic translation of languages
- Spell checking and grammar correction

Levels in NLP:



1. Lexical Analysis

- Breaking text into words (tokenization)
- Example: “NLP is fun” → [NLP, is, fun]

2.Syntactic Analysis

- Grammar and sentence structure
- Example: Parsing sentences

3.Semantic Analysis

- Understanding meaning
- Example: Word sense disambiguation

4.Discourse Analysis

- Understanding context across sentence

5.Pragmatic Analysis

- Understanding real-world meaning and intent

Applications of NLP:

- ✓ Machine Translation
- ✓ Information Retrieval
- ✓ Question Answering
- ✓ Dialogue Systems
- ✓ Information Extraction
- ✓ Entity Linking
- ✓ Text Summarization
- ✓ Text Classification
- ✓ Sentiment Analysis
- ✓ Opinion mining

Origin and Challenges in NLP:

Natural Language Processing (NLP) has evolved significantly over the decades, beginning in the mid-20th century. Its development can be divided into distinct eras, each marked by advancements in technology and methodology.

The 1950s to 1970s marked the dawn of NLP, driven by the ambition to enable machines to understand human language. Early systems were rule-based, relying on predefined linguistic rules and grammars. A notable milestone was the Georgetown-IBM experiment in 1954, which successfully translated 60 Russian sentences into English. However, these systems struggled with language complexities like idioms and context, limiting their scalability and effectiveness.

The 1980s to 1990s saw a paradigm shift with the rise of statistical methods. Machine learning algorithms replaced rigid rules, leveraging large text corpora to identify patterns and probabilities.

Techniques like n-grams and Hidden Markov Models (HMMs) became prominent, enabling applications such as speech recognition and basic machine translation. This era also introduced Named Entity Recognition (NER) and statistical machine translation, laying the groundwork for modern NLP.

The 2000s to present ushered in the deep learning revolution, transforming NLP with neural networks and advanced architectures like recurrent neural networks (RNNs) and transformers. Models such as BERT and GPT demonstrated unprecedented capabilities in understanding and generating human-like text. These advancements enabled applications like real-time translation, chatbots, and sentiment analysis. The introduction of word embeddings (e.g., Word2Vec, GloVe) further enhanced contextual understanding.

Today, NLP continues to evolve, addressing challenges like bias, contextual understanding, and multimodal integration (e.g., combining text with images or audio). Its journey from rule-based systems to AI-powered models highlights its transformative impact on human-computer interaction.

Challenges of NLP:

In this evolving landscape of artificial intelligence(AI), Natural Language Processing(NLP) stands out as an advanced technology that fills the gap between humans and machines. The ***Major Challenges of Natural language Processing(NLP)*** faced by organizations. Understanding these challenges helps you explore the advanced NLP but also leverages its capabilities to revolutionize How we interact with machines and everything from customer service automation to complicated data analysis. Natural language Processing(NLP) faces various challenges due to complexity and diversity of human languages.

1.Language Difference:

The human language and understanding is rich and intricated and there many language spoken by humans. Human language is diverse and thousand of human languages spoken around the world with having its own grammar, vocabular nuances. Human cannot understand all the languages and productivity of human language is high. There is ambiguity in natural language since same words and phrases can have different meanings and different context.

There is a **complex syntactic** structures and grammatical rules of natural languages. The rules are such as word order, verb, conjugation, tense, aspect and agreement. There is rich semantic content in human language that allows speaker to convey a wide range of meaning through words and sentences. Natural Language is pragmatics which means that how language can be used in context to approach communication goals. The human language evolves time to time with the processes such as **lexical change**.

2.Training Data

Training data is a curated collection of input-output pairs, where the input represents the features or attributes of the data, and the output is the corresponding label or target. Training data is composed of both the features (inputs) and their corresponding labels (outputs). For NLP, features might include text data, and labels could be categories, sentiments, or any other relevant annotations.

It helps the model generalize patterns from the training set to make predictions or classifications on new, previously unseen data.

3. Development Time and Resource Requirements

Development Time and Resource Requirements for *Natural Language Processing (NLP)* projects depends on various factors consisting the task complexity, size and quality of the data, availability of existing tools and libraries, and the team of expert involved. Here are some key points:

- **Complexity of the task:** Task such as classification of text or analyzing the sentiment of the text may require less time compared to more complex tasks such as machine translation or answering the questions.
- **Availability and Quality Data:** For Natural Language Processing models requires high-quality of annotated data. It can be time consuming to collect, annotate, and preprocess the large text datasets and can be resource-intensive specially for tasks that requires specialized domain knowledge or fine-tuned annotations.
- **Selection of algorithm and development of model:** It is difficult to choose the right algorithms machine learning algorithms that is best for Natural Language Processing task.
- **Evaluation and Training:** It requires powerful computation resources that consists of powerful hardware (GPUs or TPUs) and time for training the algorithms iteration. It is also important to evaluate the performance of the model with the help of suitable metrics and validation techniques for conforming the quality of the results.

4.Navigating Phrasing Ambiguities in NLP:

It is a crucial aspect to navigate phrasing ambiguities because of the inherent complexity of human languages. The cause of phrasing ambiguities is when a phrase can be evaluated in multiple ways that leads to uncertainty in understanding the meaning.The key points for navigating phrasing ambiguities in NLP:

- **Contextual Understanding:** Contextual information like previous sentences, topic focus, or conversational cues can give valuable clues for solving ambiguities.
- **Semantic Analysis:** The content of the semantic text is analyzed to find meaning based on word, lexical relationships and semantic roles. Tools such as word sense disambiguation, semantics role labelling can be helpful in solving phrasing ambiguities.
- **Syntactic Analysis:** The syntactic structure of the sentence is analyzed to find possible evaluation based on grammatical relationships and syntactic patterns.
- **Pragmatic Analysis:** Pragmatic factors such as intentions of speaker, implicatures to infer meaning of a phrase. This analysis consists of understanding the pragmatic context.
- **Statistical methods:** Statistical methods and machine learning models are used to learn patterns from data and make predictions about the input phrase.

5. Misspellings and Grammatical Errors

Overcoming Misspelling and Grammatical Error are the basic challenges in NLP, as there are different forms of linguistics noise that can impact accuracy of understanding and analysis. Here are some key points for solving misspelling and grammatical error in NLP:

- **Spell Checking:** Implement spell-check algorithms and dictionaries to find and correct misspelled words.

- **Text Normalization:** The is normalized by converting into a standard format which may contains tasks such as conversion of text to lowercase, removal of punctuation and special characters, and expanding contractions.
- **Tokenization:** The text is split into individual tokens with the help of tokenization techniques. This technique allows to identify and isolate misspelled words and grammatical error that makes it easy to correct the phrase.
- **Language Models:** With the help of language models that is trained on large corpus of data to predict the likelihood of word or phrase that is correct or not based on its context.

6. Mitigating Innate Biases in NLP Algorithms

It is a crucial step of mitigating innate biases in NLP algorithm for conforming fairness, equity, and inclusivity in natural language processing applications. Here are some key points for mitigating biases in NLP algorithms.

- **Collection of data and annotation:** It is very important to confirm that the training data used to develop NLP algorithms is diverse, representative and free from biases.
- **Analysis and Detection of bias**: Apply **bias detection** and analysis method on training data to find biases that is based on demographic factors such as race, gender, age.
- **Data Preprocessing:** Data Preprocessing the most important process to train data to mitigate biases like debiasing word embeddings, balance class distributions and augmenting underrepresented samples.
- **Fair representation learning:** Natural Language Processing models are trained to learn fair representations that are invariant to protect attributes like race or gender.
- **Auditing and Evaluation of Models:** Natural Language models are evaluated for fairness and bias with the help of metrics and audits. NLP models are evaluated on diverse datasets and perform post-hoc analyses to find and mitigate innate biases in NLP algorithms.

7. Words with Multiple Meanings

Words with multiple meaning plays a lexical challenge in ***Nature Language Processing*** because of the ambiguity of the word. These words with multiple meaning are known as polysemous or homonymous have different meaning based on the context in which they are used. Here are some key points for representing the lexical challenge plays by words with multiple meanings in NLP:

- **Semantic analysis:** Implement semantic analysis techniques to find the underlying meaning of the word in various contexts. Word embedding or semantic networks are the semantic representation can find the semantic similarity and relatedness between different word sense.
- **Domain specific knowledge:** It is very important to have a specific domain-knowledge in Natural Processing tasks that can be helpful in providing valuable context and constraints for determining the correct context of the word.
- **Multi-word Expression (MWEs):** The meaning of the entire sentence or phrase is analyzed to disambiguate the word with multiple meanings.
- **Knowledge Graphs and Ontologies:** Apply knowledge graphs and ontologies to find the semantic relationships between different words context.

8. Addressing Multilingualism

It is very important to address language diversity and multilingualism in Natural Language Processing to confirm that NLP systems can handle the text data in multiple languages effectively. Here are some key points to address language diversity and multilingualism:

- **Multilingual Corpora:** Multilingual corpus consists of text data in various languages and serve as valuable resources for training NLP models and systems.
- **Cross-Lingual Transfer Learning:** This is a type of techniques that is used to transfer knowledge learned from one language to another.

- **Language Identification:** Design language identification models to automatically detect the language of a given text.
- **Machine Translation:** Machine Translation provides the facility to communicate and inform access across language barriers and can be used as preprocessing step for multilingual NLP tasks.

9. Reducing Uncertainty and False Positives in NLP

It is very crucial task to reduce uncertainty and false positives in Natural Language Process (NLP) to improve the accuracy and reliability of the NLP models. Here are some key points to approach the solution:

- **Probabilistic Models**: Use probabilistic models to figure out the uncertainty in predictions. Probabilistic models such as Bayesian networks gives probabilistic estimates of outputs that allow uncertainty quantification and better decision making.
- **Confidence Scores:** The confidence scores or probability estimates is calculated for NLP predictions to assess the certainty of the output of the model. Confidence scores helps us to identify cases where the model is uncertain or likely to produce false positives.
- **Threshold Tuning:** For the classification tasks the decision thresholds is adjusted to make the balance between sensitivity (recall) and specificity. False Positives in NLP can be reduced by setting the appropriate thresholds.
- **Ensemble Methods:** Apply ensemble learning techniques to join multiple model to reduce uncertainty.

10. Facilitating Continuous Conversations with NLP

Facilitating continuous conversations with NLP includes the development of system that understands and responds to human language in real-time that enables seamless interaction between users and machines. Implementing real time natural language processing pipelines gives to capability to analyze and interpret user input as it is received involving algorithms are optimized and systems for low latency processing to confirm quick responses to user queries and inputs.

Building an NLP models that can maintain the context throughout a conversation. The understanding of context enables systems to interpret user intent, conversation history tracking, and generating relevant responses based on the ongoing dialogue. Apply intent recognition algorithm to find the underlying goals and intentions expressed by users in their messages.

Language and Grammar in NLP:

Language: Natural Language Processing (NLP) is a branch of Artificial Intelligence that focuses on the interaction between computers and human language. In NLP, *language* refers to natural human languages such as English, Telugu, Hindi, Tamil, etc., which are used for communication in spoken or written form. The main aim of NLP is to enable computers to understand, interpret, and generate human language in a meaningful way. Language in NLP can be categorized into natural language and formal language, but NLP primarily works with natural language.

Natural Language Processing (NLP) analyzes human language at **different levels** to understand its structure and meaning. Each level focuses on a specific aspect of language, starting from sounds to complete discourse. These levels work together to help computers understand and generate natural language effectively.

1. Phonological Level (Sound Level)

- Deals with **speech sounds** and their patterns.

- Important for **spoken language processing**.
- Helps in recognizing how sounds combine to form words.
- Used mainly in **speech recognition** and **text-to-speech systems**.

Example:
Distinguishing between sounds like */b/* and */p/* in spoken words.

2. Morphological Level (Word Formation)

- Studies the **internal structure of words**.
- Words are made of **morphemes** (smallest meaningful units).
- Helps in understanding tense, number, gender, and derivation.

Example:

- *play → playing → played*
- *unhappy = un + happy*

3. Lexical Level (Word Meaning)

- Focuses on **individual words and their meanings**.
- Uses dictionaries or lexicons to identify word categories (noun, verb, etc.).
- Helps in handling **polysemy** (one word, multiple meanings).

Example:

- *bank* → financial bank / river bank

4. Syntactic Level (Grammar)

- Deals with **sentence structure and grammar rules**.
- Determines whether a sentence is grammatically correct.
- Identifies relationships between words in a sentence.

Example:

- Correct: *She is reading a book.*
- Incorrect: *She reading is book.*

5. Semantic Level (Meaning)

- Concerned with **meaning of words and sentences**.
- Resolves ambiguity using context.
- Helps machines understand what a sentence actually means.

Example:

- *"I saw a man with a telescope."* (Who has the telescope?)

6. Pragmatic Level (Context)

- Interprets meaning based on **situation and intent**.
- Considers speaker's intention rather than literal meaning.

Example:

- *"Can you open the window?"* → a request, not a question about ability.

7. Discourse Level (Conversation)

- Deals with **relationships between multiple sentences**.
- Helps in understanding references like pronouns.
- Important for conversation and text coherence.

Example:

- *Ravi went home. He was tired.*
  (*He* refers to *Ravi*)

Applications of Languages in NLP:

1. Machine Translation
2. Speech Recognition
3. Text-to-Speech
4. Chatbots & Virtual Assistants
5. Sentiment Analysis
6. Information Retrieval
7. Text Summarization
8. Named Entity Recognition (NER)
9. Spell & Grammar Checking
10. Question Answering Systems

Grammar in NLP:

Grammar in Natural Language Processing (NLP) refers to the set of rules that define the structure of sentences in a natural language. These rules help a computer understand how words are arranged to form meaningful sentences. Grammar is essential in NLP because it enables machines to analyze sentence structure, identify relationships between words, and interpret meaning correctly.

Types of Grammar in NLP:

1. Transformational Grammar
2. Lexical Functional Grammar (LFG)
3. Government and Binding Grammar (GB)
4. Generalized Phrase Structure Grammar (GPSG)
5. Dependency Grammar (DG)
6. Paninian Grammar (PG)
7. Tree-Adjoining Grammar (TAG)
8. Context-Free Grammar (CFG)

9. Probabilistic Grammar

1. Transformational Grammar:

Transformational Grammar, also known as **Transformational–Generative Grammar**, is a linguistic theory proposed by **Noam Chomsky**. It explains how sentences in a language are generated and transformed from basic structures into more complex forms. This grammar plays an important role in **Natural Language Processing (NLP)** for understanding deep sentence structure and meaning.

Transformational grammar is based on the idea that every sentence has two levels of structure: **Deep Structure** and **Surface Structure**. The **deep structure** represents the core meaning of a sentence, while the **surface structure** represents the actual spoken or written form. Transformational rules convert deep structures into surface structures.

The grammar consists of two main components: **Phrase Structure Rules** and **Transformational Rules**. Phrase structure rules generate basic sentence patterns such as $S \rightarrow NP + VP$. Transformational rules modify these basic structures to form different types of sentences like passive, interrogative, or negative sentences.

For example:

- Active: *The teacher explains the lesson.*
- Passive (transformation): *The lesson is explained by the teacher.*

Both sentences have similar deep meaning but different surface forms.

2.Lexical Functional Grammar (LFG):

Lexical Functional Grammar (LFG) is a theory of grammar proposed by **Joan Bresnan and Ronald Kaplan**. It focuses on the relationship between **words (lexicon)** and their **grammatical functions** in a sentence. LFG explains sentence structure without using transformational rules.

LFG represents sentences using two main structures:

- **C-structure (Constituent structure):** Shows phrase structure (word order).
- **F-structure (Functional structure):** Shows grammatical functions like subject, object, tense, etc.

Example:

Sentence: *She reads a book.*

- **C-structure:**
  NP (She) + VP (reads a book)
- **F-structure:**
  - Subject: She
  - Predicate: read
  - Object: book
  - Tense: Present

3.Government and Binding Grammar:

Government and Binding (GB) Grammar is a linguistic theory proposed by **Noam Chomsky**. It explains sentence structure using a set of general principles instead of many specific rules. GB grammar aims to describe how different languages share common grammatical principles.

GB grammar has two important components: **Government** and **Binding**.

- **Government** deals with the relationship between a head (like a verb or preposition) and its complements, helping to assign grammatical roles such as case.
- **Binding** explains how pronouns and noun phrases refer to each other in a sentence.

Example:

Sentence: *Ravi said that he will come.*

- **Binding:** The pronoun **"he"** refers to **"Ravi"**.
- **Government:** The verb **"said"** governs the clause *that he will come*.

Government and Binding Grammar is important in NLP for **sentence parsing**, **pronoun resolution**, and **syntactic analysis**.

4.Generalized Phase Structure Grammar (GPSG):

Generalized Phrase Structure Grammar (GPSG) is a syntactic theory developed by **Gerald Gazdar and others**. It is a **non-transformational grammar**, meaning it explains sentence structure without using transformational rules. Instead, it uses a set of formal phrase structure rules and constraints.

GPSG represents sentences using **phrase structure rules**, **feature systems**, and **principles** that restrict how rules apply. It focuses on describing grammar in a highly **formal and computationally suitable** way, making it useful for **Natural Language Processing (NLP)**.

Example:

Sentence: *The girl eats an apple.*

Phrase structure representation:

- S → NP + VP
- NP → Det + N (*The girl*)
- VP → V + NP (*eats an apple*)

GPSG helps NLP systems in **syntactic parsing**, **grammar checking**, and **language modeling** by providing a clear and rule-based description of sentence structure.

5.Dependency Grammar (DG):

Dependency Grammar is a grammatical theory that represents sentence structure based on **dependency relations between words**. Instead of phrase structures, it focuses on how words depend on one another, especially on a **head word** (usually the verb).

In dependency grammar, every word (except the root) depends on another word, and these relationships show grammatical functions like subject, object, and modifier. This approach is simple and well-suited for **Natural Language Processing (NLP)** tasks such as parsing and information extraction.

Example:

Sentence: *The boy reads a book.*

- **Head (root):** reads
- **Subject:** boy → depends on *reads*
- **Object:** book → depends on *reads*
- **Determiners:** the → boy, a → book

Thus, dependency grammar clearly shows word relationships, making it useful for **syntactic analysis**, **machine translation**, and **question answering systems** in NLP.

6.Paninian Grammar (PG):

Paninian Grammar is an ancient Indian grammatical framework developed by Panini for Sanskrit. It is a dependency-based grammar that focuses on the relationship between words rather than word order. In Natural Language Processing (NLP), Paninian grammar is especially useful for Indian languages like Sanskrit, Hindi, Telugu, and other free word–order languages.

The core concept of Paninian grammar is the Karaka system, which defines the semantic roles of nouns in a sentence, such as doer (Karta), object (Karma), instrument (Karana), location (Adhikarana), etc. These roles help identify meaning even when word order changes.

Example:

Sentence:
Ram ate fruit

- Karta (doer): Ram
- Karma (object): fruit
- Kriya (verb): ate

Even if the word order changes, the meaning remains clear due to Karaka relations.

Thus, Paninian grammar provides a powerful model for NLP applications like machine translation, parsing, and semantic analysis of Indian languages.

7.Tree Adjoining Grammar:

Tree Adjoining Grammar (TAG) is a formal grammar framework used in **Natural Language Processing (NLP)** to describe the syntactic structure of sentences. It was introduced by **Aravind Joshi**. TAG represents grammar using **trees** instead of simple rewrite rules.

In TAG, grammar is defined using two types of elementary trees:

- **Initial trees** – represent basic sentence structures

- **Auxiliary trees** – allow recursion and modification through an operation called **adjunction**

The main operations in TAG are **substitution** and **adjunction**, which combine trees to form complex sentences.

Example:

Sentence: *She is reading a book.*

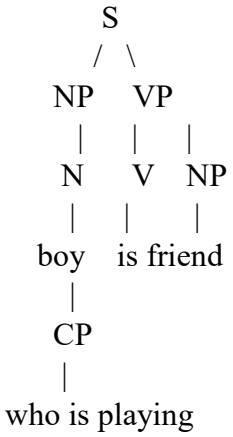
- An **initial tree** represents the basic structure: *She reads a book*
- An **auxiliary tree** adjoins to add tense/aspect: *is reading*

By adjoining the auxiliary tree to the initial tree, the complete sentence is formed.

Tree Adjoining Grammar is powerful for handling **long-distance dependencies** and is useful in NLP tasks such as **parsing**, **machine translation**, and **syntactic analysis**.

Example:

The boy who is playing is my friend.



8.Context Free Grammar (CFG):

Context-Free Grammar (CFG) is a formal grammar used in **Natural Language Processing (NLP)** to describe the syntactic structure of languages. In CFG, production rules define how sentences are formed, and each rule can be applied **independently of context**.

A CFG consists of **four components**:

1. **Non-terminals** (S, NP, VP)
2. **Terminals** (actual words)
3. **Production rules**
4. **Start symbol** (usually S)

CFG is widely used for **sentence parsing** and **syntax analysis** in NLP.

Example:

Sentence: *The boy eats an apple.*

Grammar rules:

- S → NP VP
- NP → Det N
- VP → V NP
- Det → the, an
- N → boy, apple
- V → eats

Using these rules, the sentence structure can be generated and analyzed.

Thus, Context-Free Grammar provides a clear and rule-based way to model sentence structure and is essential in many NLP applications like **parsers** and **language translators**.

9.Probabilistic Grammar:

Probabilistic Grammar, also called **Probabilistic Context-Free Grammar (PCFG)**, is an extension of Context-Free Grammar used in **Natural Language Processing (NLP)**. In this grammar, each production rule is assigned a **probability**, which helps the system choose the most likely parse when a sentence has multiple interpretations.

Probabilistic grammar is useful for handling **ambiguity** in natural language. The sum of probabilities of all rules with the same left-hand side equals **1**. During parsing, the parse tree with the **highest probability** is selected as the correct one.

Example:

Grammar rules with probabilities:

- S → NP VP (1.0)
- NP → Det N (0.6)
- NP → NP PP (0.4)
- VP → V NP (0.7)
- VP → VP PP (0.3)

Sentence: *I saw the man with a telescope.*

Probabilistic grammar helps decide whether *"with a telescope"* modifies **saw** or **man** based on higher probability.

Thus, probabilistic grammar improves parsing accuracy and is widely used in **speech recognition**, **machine translation**, and **syntactic parsing** in NLP.

Regular Expression and Finite State Automata:

Regular Expression:

- Helps in **understanding sentence structure**

- Identifies **subject, verb, object**, and modifiers
- Reduces **ambiguity** in language interpretation
- Supports **syntactic parsing** and analysis
- Improves **machine translation** accuracy
- Essential for **speech recognition** systems
- Enables correct **text generation**
- Helps in **question answering** and chatbots
- Maintains **meaning and grammatical correctness**
- Core component of **Natural Language Processing**

A **Regular Expression (Regex)** is a sequence of characters used to define a **pattern** for searching and manipulating text. In **Natural Language Processing (NLP)**, regular expressions are mainly used during **text preprocessing** to identify and handle specific text patterns.

Regular expressions help in tasks such as **tokenization**, **removing unwanted characters**, **extracting useful information** like phone numbers, email IDs, and dates, and **pattern matching**. They make text data clean and structured before applying NLP algorithms.

Finite State Automata:

Finite State Automata (FSA) is a mathematical model used to recognize patterns and process strings of symbols. In **Natural Language Processing (NLP)**, FSA is widely used for **lexical analysis**, **tokenization**, **morphological analysis**, and **pattern matching**.

A **Finite State Automaton** is a machine that consists of a **finite number of states** and processes input symbols **one at a time**, moving from one state to another according to transition rules.

Components of Finite State Automata:

An FSA is defined by **five elements**:

1. **Q** – A finite set of states
2. **Σ (Sigma)** – Input alphabet (set of symbols)
3. **δ (Delta)** – Transition function
4. $\mathbf{q_0}$ – Start (initial) state
5. **F** – Set of final (accepting) states

Types of Finite State Automata:

1. Deterministic Finite State Automata
2. Non-Deterministic Finite State Automata

1.Deterministic Finite State Automata:

A **Deterministic Finite Automaton (DFA)** is a type of **Finite State Automaton** used to recognize patterns in strings. It is called *deterministic* because **for every state and input symbol, there is exactly one possible next state**. DFA is widely used in **theory of computation** and **Natural Language Processing (NLP)** for tasks like **lexical analysis, tokenization, and pattern matching**.

A DFA is defined by a 5-tuple:

DFA = (Q, Σ, δ, q_0, F)

Where:

- **Q** – Finite set of states
- **Σ (Sigma)** – Input alphabet
- **δ (Delta)** – Transition function ($Q \times \Sigma \rightarrow Q$)
- **q_0** – Initial state
- **F** – Set of final (accepting) states

Characteristics of DFA

- Exactly **one transition** for each input symbol from a state
- **No ε (empty) transitions**
- Always knows **which state to go next**
- Simple and predictable behaviour

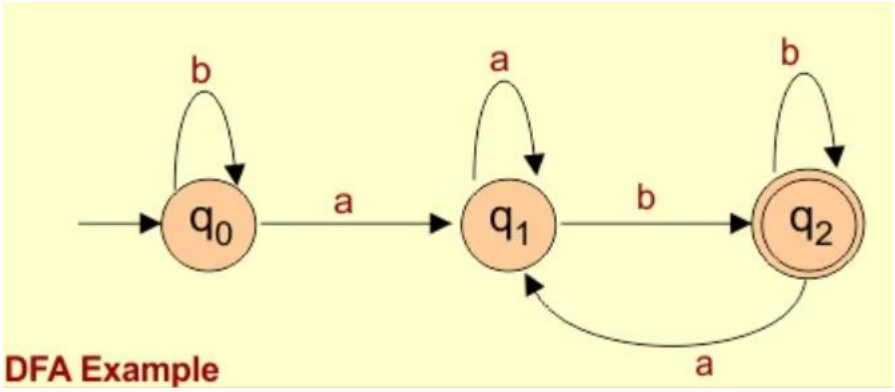
Working of DFA

1. Start at the **initial state (q_0)**
2. Read input symbols **one by one**
3. Move to the next state using the transition function
4. After all input is read:
    - If the current state $\in$ F $\rightarrow$ **Accepted**
    - Else $\rightarrow$ **Rejected**

Example:

All strings ending with b

Alphabets= (a,b)



2.Non-Deterministic Finite State Automata:

A **Non-Deterministic Finite Automaton (NFA)** is a type of **Finite State Automaton (FSA)** in which **more than one transition is possible** for a given state and input symbol. Unlike DFA, an NFA can move to **multiple next states** or even change states **without consuming any input** using **ε (epsilon) transitions**. NFAs are important in **theory of computation** and **Natural Language Processing (NLP)**, especially in **pattern matching and regular expression processing**.

An NFA is defined as a 5-tuple:

NFA = (Q, Σ, δ, q_0, F)

Where:

- **Q** – Finite set of states
- **Σ (Sigma)** – Input alphabet
- **δ (Delta)** – Transition function
    - $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(Q)$
- **q_0** – Initial state
- **F** – Set of final (accepting) states

Characteristics of NFA

- Multiple transitions may exist for the same input symbol
- **ε-transitions** are allowed
- The automaton can be in **multiple states at the same time**
- Acceptance occurs if **any path** reaches a final state

Working of NFA

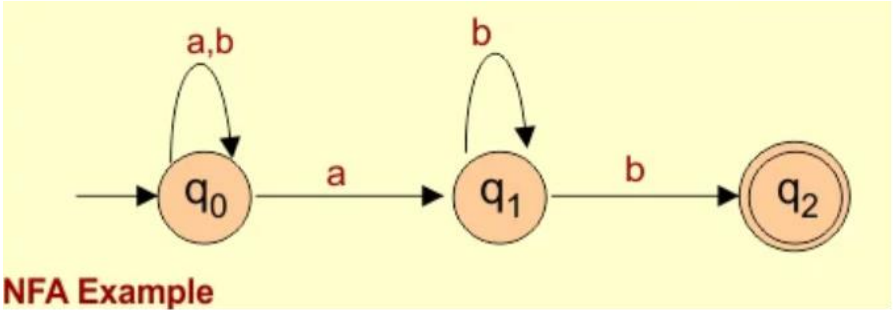
1. Start from the **initial state**
2. Read input symbols (or make ε-moves)
3. Follow **all possible transitions simultaneously**

4. If **at least one computation path** ends in a final state after consuming the input, the string is **accepted**

Example:

All strings that ends with “b”

Alphabets= (a,b)



Tokenization:

- Tokenization is the process of **breaking a large text into smaller, meaningful units called tokens**.
- These tokens are the **building blocks for NLP tasks**.
- Tokenization is often the **first preprocessing step** in NLP.

Token: A unit of text, usually a **word, subword, or character**, depending on the tokenization method.

The primary goal is to segment text into meaningful units, which are often individual words, to facilitate further analysis such as text classification, sentiment analysis, machine translation, or information retrieval.

Types of Tokenization:

1. Word Tokenization
2. Sentence Tokenization
3. Character Tokenization
4. Subword Tokenization
5. N-gram Tokenization

1.Word Tokenization:

Word tokenization is the process of **dividing a text into individual words**, known as **tokens**. These tokens act as the **basic units of text analysis** in Natural Language Processing (NLP).

In natural language, a sentence is a continuous sequence of characters. NLP systems cannot directly understand this raw text. Word tokenization converts the text into a **structured form** by identifying **word boundaries**, usually based on **spaces and punctuation marks**.

Example

Sentence:

NLP is fun, isn't it?

After word tokenization, the text is divided into tokens such as:

NLP | is | fun | , | is | n’t | it | ?

Importance of Word Tokenization:

Word tokenization is important because:

1. It is the **first preprocessing step** in NLP.
2. It converts unstructured text into **meaningful units**.
3. It enables **word frequency counting** and statistical analysis.
4. It supports downstream NLP tasks such as:
    - Text classification
    - Sentiment analysis
    - Machine translation
5. It helps in **feature extraction** for machine learning models.

Challenges in Word Tokenization:

1. **Punctuation handling** – deciding whether punctuation should be a separate token.
2. **Contractions** – words like *don’t*, *isn’t* can be split in multiple ways.
3. **Hyphenated words** – *well-being*, *state-of-the-art*.
4. **Multi-word expressions** – proper nouns like *New York*.
5. **Language dependency** – some languages do not use spaces between words.

Applications of Word Tokenization:

- Text mining
- Information retrieval
- Sentiment analysis
- Machine translation
- Chatbots and virtual assistants
- Search engines

2.Sentence Tokenization:

Sentence tokenization is the process of **dividing a text into individual sentences**, called **sentence tokens**. It helps NLP systems understand the **sentence boundaries** within a document.

A paragraph or document usually contains multiple sentences. For effective language processing, NLP systems must first identify where **one sentence ends and another begins**. Sentence tokenization detects these boundaries using **punctuation marks** such as full stops (.), question marks (?), and exclamation marks (!), along with linguistic rules.

Example:

Text:

NLP is fun. It is widely used in AI. Is it difficult?

After sentence tokenization, the text is split into:

1. NLP is fun.
2. It is widely used in AI.
3. Is it difficult?

Importance of Sentence Tokenization

Sentence tokenization is important because:

1. It is a **basic preprocessing step** in NLP.
2. It helps in understanding the **structure and meaning** of text.
3. It improves the performance of **parsing and syntactic analysis**.
4. It is required for applications such as:
    - Machine translation
    - Text summarization
    - Question answering
    - Speech processing

Challenges in Sentence Tokenization

1. **Abbreviations** – *Dr. Smith*, *etc.*, *U.S.A.*
2. **Decimal numbers** – *3.14* should not end a sentence.
3. **Ellipsis** – *Wait... what happened?*
4. **Quotations** – sentences ending inside quotes.
5. **Language dependency** – different languages follow different punctuation rules.

Applications of Sentence Tokenization

- Text summarization
- Information extraction
- Dialogue systems

- Speech recognition
- Document analysis

3.Character Tokenization:

Character tokenization is the process of breaking text into individual characters, where each character is treated as a token. Unlike word or sentence tokenization, the smallest unit of text is considered.

In character tokenization, a text string is viewed as a **sequence of characters** rather than words or sentences. This includes **letters, digits, punctuation marks, and spaces**. Each character is processed independently by the NLP system. This method does not rely on spaces or word boundaries, making it suitable for languages and tasks where word segmentation is difficult.

Example

Text:

NLP

After character tokenization:

N | L | P

Another example:

AI is fun

Tokens:

A | I | (space) | i | s | (space) | f | u | n

Importance of Character Tokenization:

Character tokenization is important because:

1. It avoids **language-dependent rules**.
2. It handles **unknown or rare words** naturally.
3. It is effective for **morphologically rich languages**.
4. It supports **fine-grained text analysis**.

Challenges in Character Tokenization:

1. Large sequence length – produces many tokens.
2. Loss of word-level meaning.
3. Higher computational cost.
4. Difficulty in capturing semantic context.

Applications of Character Tokenization:

- Spelling correction

- Text normalization
- Language identification
- Named Entity Recognition (NER)
- Handling noisy text (social media data)

4.Sub-Word Tokenization:

Sub-word tokenization is the process of **dividing words into smaller meaningful units called sub-words**. These sub-words may be complete words, prefixes, suffixes, or frequently occurring character sequences.

In traditional word tokenization, each word is treated as a single token. However, this causes problems when **rare, new, or unknown words** appear. Sub-word tokenization solves this by breaking words into **frequently occurring sub-units**, allowing models to understand and process unseen words.

Example

Word:

unhappiness

Sub-word tokens:

un | happi | ness

Another example:

playing

Sub-word tokens:

play | ing

Importance of Sub-word Tokenization

Sub-word tokenization is important because:

1. It handles **out-of-vocabulary (OOV) words** effectively.
2. It reduces the **size of the vocabulary**.
3. It captures **morphological information** (prefixes and suffixes).
4. It improves performance of **deep learning models**.
5. It is widely used in **modern NLP models** such as BERT, GPT, and T5.

Challenges in Sub-word Tokenization:

1. Increased **sequence length** compared to word tokens.
2. Loss of complete word-level semantics.
3. Requires **training a tokenizer** on large corpora.
4. More complex than simple word tokenization.

Applications of Sub-word Tokenization:

- Language modeling
- Machine translation
- Text generation
- Question answering
- Speech recognition

5.N-grams Tokenization:

N-gram tokenization is a technique in which a text is divided into **contiguous sequences of *n* items**, called **n-grams**.
The items can be **characters, syllables, or words**, depending on the application.

- **Unigram** → n = 1
- **Bigram** → n = 2
- **Trigram** → n=3

Instead of treating a sentence as a collection of independent words, n-gram tokenization captures **local context** by grouping adjacent tokens together. This helps in understanding **word order and relationships** between neighboring tokens.N-grams are commonly generated **after basic tokenization** (word or character level).

Example (Word-level N-grams)

Sentence:

NLP is fun

- **Unigrams (n=1):**
  NLP | is | fun
- **Bigrams (n=2):**
  NLP is | is fun
- **Trigrams (n=3):**
  NLP is fun

Importance of N-gram Tokenization:

N-gram tokenization is important because:

1. It captures **context and word order**.
2. It improves **language modeling**.
3. It helps in **predicting the next word**.
4. It is useful in statistical NLP and classical ML models.
5. It enhances feature representation for text classification.

Challenges of N-gram Tokenization

1. **Data sparsity** – higher n values create many rare n-grams.
2. **Large feature space** – memory and computation increase.
3. **Limited context** – only captures local dependencies.
4. **Not suitable for long-range semantics**.

Applications of N-gram Tokenization:

- Language modeling
- Text classification
- Spell checking
- Machine translation
- Speech recognition
- Plagiarism detection

Text Segmentation:

Text segmentation is the process of **dividing a continuous text into meaningful units**, such as **sentences, words, topics, or discourse segments**, to make the text easier to analyze and process in Natural Language Processing (NLP).

Natural language text is often long and unstructured. NLP systems cannot process it effectively as a whole. Text segmentation breaks the text into **smaller, logically coherent segments**, allowing better understanding, analysis, and interpretation.

Segmentation can occur at **multiple levels**, ranging from characters to paragraphs and topics.

Importance of Text Segmentation:

Text segmentation is important because:

1. It improves **text understanding**.
2. It reduces **computational complexity**.
3. It enables accurate **feature extraction**.
4. It enhances performance of downstream NLP tasks.
5. It is essential for processing **long documents**.

Levels of Text Segmentation:

(Input) Raw Text

↓

Paragraph Segmentation

↓

Sentence Segmentation

↓

Word Segmentation

↓

Sub-word Segmentation

↓

Character Segmentation

1.Paragraph Segmentation:

Paragraph segmentation is the process of **dividing a continuous text into coherent paragraphs**, where each paragraph represents a **group of sentences discussing a single idea or topic**.

In natural language text, paragraphs organize content into **logical units**. Paragraph segmentation identifies **paragraph boundaries** to separate different ideas, arguments, or topics. This segmentation helps NLP systems process and analyze text in a structured manner.

Unlike sentence segmentation, paragraph segmentation focuses on **semantic coherence** rather than punctuation alone.

Example

Text:

NLP is a field of artificial intelligence that deals with language. It enables computers to understand human language.
It is widely used in chatbots and translators. These applications are common today.

After paragraph segmentation:

paragraph 1:

NLP is a field of artificial intelligence that deals with language. It enables computers to understand human language.

paragraph 2:

It is widely used in chatbots and translators. These applications are common today.

Importance of Paragraph Segmentation

Paragraph segmentation is important because:

1. It improves **document structure understanding**.
2. It supports **topic identification and discourse analysis**.
3. It enhances performance of:
    - Text summarization
    - Information retrieval

- Document classification

4. It helps NLP systems process **long texts efficiently**.
5. It preserves **semantic coherence**.

Challenges in Paragraph Segmentation

1. **Lack of clear boundaries** in plain text.
2. **Topic overlap** between adjacent paragraphs.
3. **Inconsistent formatting**.
4. **Language dependency**.
5. **Short paragraphs** with weak signals.

Applications of Paragraph Segmentation

- Document summarization
- Topic modeling
- Information extraction
- Question answering
- Educational content analysis

2.Sentence Segmentation:

Sentence segmentation is the process of **dividing a continuous text into individual sentences** by identifying **sentence boundaries**. Each sentence is treated as a **separate and meaningful unit** for further Natural Language Processing (NLP) tasks.

Natural language text often consists of multiple sentences written without explicit markers other than punctuation. Sentence segmentation determines where a sentence **starts and ends** using punctuation marks, linguistic rules, and contextual information.

Accurate sentence segmentation is essential because many NLP tasks operate **at the sentence level**.

Example

Text:

NLP is a part of artificial intelligence. It helps machines understand human language. Is it useful?

After sentence segmentation:

1. NLP is a part of artificial intelligence.
2. It helps machines understand human language.
3. Is it useful?

Importance of Sentence Segmentation

Sentence segmentation is important because:

1. It is a **basic preprocessing step** in NLP.
2. It improves **syntactic and semantic analysis**.
3. It enhances accuracy of:
    - Machine translation
    - Text summarization
    - Question answering
4. It helps NLP models understand **text structure**.
5. It reduces errors in downstream processing.

Challenges in Sentence Segmentation

1. **Abbreviations** – *Dr.*, *etc.*, *U.S.A.*
2. **Decimal numbers** – *3.14*.
3. **Ellipses** – *Wait...*
4. **Quotes and parentheses**.
5. **Informal text** such as social media content.

Applications of Sentence Segmentation

- Text summarization
- Machine translation
- Information extraction
- Dialogue systems
- Speech processing

3.Word Segmentation:

Word segmentation is the process of **dividing a continuous sequence of text into individual words**. It identifies **word boundaries**, especially in texts where **spaces are absent or unreliable**.

In some languages (like English), spaces clearly separate words. However, in many languages such as **Chinese, Japanese, Thai**, words are written **without spaces**. Word segmentation determines where one word ends and another begins.

Word segmentation is different from simple word tokenization because it often requires **linguistic, statistical, or semantic analysis** to correctly identify words.

Example

English (with spaces)

Text:

Natural Language Processing

After word segmentation:

Natural | Language | Processing

Importance of Word Segmentation

Word segmentation is important because:

1. It enables **lexical analysis**.
2. It is essential for **languages without whitespace**.
3. It improves accuracy of:
    - Machine translation
    - Information retrieval
    - Speech recognition
4. It supports feature extraction and parsing.
5. It is a foundation for higher-level NLP tasks.

Challenges in Word Segmentation

1. **Ambiguity** – multiple valid segmentations.
2. **Unknown words** – new terms or names.
3. **Compound words**.
4. **Language dependency**.
5. **Noisy or informal text**.

Applications of Word Segmentation

- Machine translation
- Search engines
- Text summarization
- Speech recognition
- Named Entity Recognition

4.Sub-Word Segmentation:

Subword segmentation is the process of **dividing words into smaller units called subwords**, which may be meaningful parts such as **prefixes, suffixes, or frequently occurring character sequences**. It lies between **word-level** and **character-level** processing.

In traditional word segmentation, each word is treated as a single unit. This leads to problems with **rare, unseen, or newly formed words**. Subword segmentation overcomes this by representing words

as **combinations of smaller known units**, enabling NLP systems to handle unknown vocabulary efficiently.

Subwords allow a balance between capturing **semantic meaning** and reducing **vocabulary size**.

Example

Word:

unhappiness

After subword segmentation:

un | happi | ness

Another example:

internationalization

Subwords:

inter | nation | al | ization

Importance of Subword Segmentation

Subword segmentation is important because:

1. It effectively handles **out-of-vocabulary (OOV) words**.
2. It reduces the **size of the vocabulary**.
3. It captures **morphological structure** of words.
4. It improves generalization of NLP models.
5. It is widely used in **modern deep learning models**.

Challenges of Subword Segmentation

1. Increased sequence length compared to word tokens.
2. Partial loss of word-level semantics.
3. Complexity in training the tokenizer.
4. Less interpretable subwords.

Applications of Subword Segmentation

- Language modeling
- Machine translation
- Text generation
- Question answering
- Speech recognition

5.Character Segmentation:

Character segmentation is the process of **dividing text into individual characters**, where **each character is treated as a separate unit** for analysis in Natural Language Processing (NLP).

In character segmentation, text is viewed as a **sequence of characters rather than words or sentences**. These characters may include **letters, digits, punctuation marks, and spaces**. This approach does not depend on word boundaries or spaces, making it language-independent.

Character segmentation is especially useful when word boundaries are unclear or absent.

Example

Text:

NLP

After character segmentation:

N | L | P

Another example:

AI is fun

Character segments:

A | I | (space) | i | s | (space) | f | u | n

Importance of Character Segmentation

Character segmentation is important because:

1. It avoids dependency on **language-specific rules**.
2. It naturally handles **unknown or rare words**.
3. It is effective for **morphologically complex languages**.
4. It enables fine-grained text analysis.

Challenges of Character Segmentation

1. Produces very long sequences.
2. Higher computational cost.
3. Limited semantic information per token.
4. Difficult to capture long-range meaning.

Applications of Character Segmentation

- Spelling correction
- Language identification
- Named Entity Recognition

- Text normalization
- Processing social media text

Sentence Splitting:

Sentence Splitting, also known as **Sentence Segmentation** or **Sentence Boundary Detection (SBD)**, is the process of dividing a continuous text into **individual sentences**. It is an important **preprocessing step** in Natural Language Processing (NLP).

Importance of Sentence Splitting

Sentence splitting is essential because:

- Most NLP tasks operate **sentence-wise**
- Helps preserve **syntactic and semantic meaning**
- Improves performance of:
    - Tokenization
    - POS tagging
    - Parsing
    - Machine Translation
    - Sentiment Analysis
    - Text Summarization

Incorrect sentence splitting can lead to **loss of meaning**.

Challenges in Sentence Splitting

Sentence splitting is difficult due to:

- **Abbreviations**: Dr., Mr., etc.
- **Decimals**: 3.14
- **Initials**: A.P.J. Abdul Kalam
- **URLs and emails**
- **Quotation marks**

Example:

Dr. Rao teaches NLP at 10.30 a.m. It is interesting.

Example of Sentence Splitting

Input Text:

NLP is a part of AI. Dr. Rao teaches NLP at 10.30 a.m. It is useful!

Output Sentences:

1. NLP is a part of AI.
2. Dr. Rao teaches NLP at 10.30 a.m.
3. It is useful!

Python Code for Sentence Splitting (Using NLTK)

```
import nltk
nltk.download('punkt')
from nltk.tokenize import sent_tokenize
text = "NLP is a part of AI. Dr. Rao teaches NLP at 10.30 a.m. It is useful!"
sentences = sent_tokenize(text)
for s in sentences:
    print(s)
```

Output

NLP is a part of AI.

Dr. Rao teaches NLP at 10.30 a.m.

It is useful!

Applications of Sentence Splitting

- Chatbots
- Question Answering Systems
- Speech Recognition
- Information Extraction
- Machine Translation
- Text Summarization

Morphological Parsing in NLP:

Morphological Parsing is the process of **analyzing the internal structure of words** to identify their **root (stem)** and **affixes** such as prefixes and suffixes. It helps in understanding **how words are formed and how their meanings change**. A **morpheme** is the smallest meaningful unit of a word. **Example:**

- *unhappiness* = **un + happy + ness**

Morphological parsing is important because it:

- Reduces vocabulary size
- Improves machine translation and information retrieval

- Helps in POS tagging and syntactic analysis

Example of morphological parsing:

- *running* → run + ing
- *teachers* → teach + er + s

Stemming and Lemmatization in NLP:

Stemming is a text normalization technique in **Natural Language Processing (NLP)** that reduces words to their **base or root form**, called a **stem**, by removing prefixes or suffixes. The resulting stem **may not be a valid dictionary word**.

Stemming is required to:

- Reduce **vocabulary size**
- Treat related words as the **same term**
- Improve **search and information retrieval**
- Improve performance of **text classification, clustering, and sentiment analysis**
- Reduce computational cost

Stemming works by applying **rule-based heuristics** that:

- Remove common suffixes such as -ing, -ed, -s, -ly
- Sometimes remove prefixes

It does **not consider context or meaning**, only word form.

Example:

- playing, played, plays → **play**
- studies → **studi**

Example:

Original word	**Stemming**
Playing	Play
Played	Play
Plays	Play
Happiness	Happi

Connection	Connection
Studies	Studi

Lemmatization:

Lemmatization is a crucial text preprocessing technique in **Natural Language Processing (NLP)** that reduces words to their base or dictionary form, known as the **lemma**. Unlike stemming, which often removes suffixes without considering the word's meaning, lemmatization ensures the resulting word is meaningful and exists in the language's dictionary. For example, "running" becomes "run," and "better" becomes "good."

Lemmatization relies on linguistic rules and dictionaries to determine the base form of a word. It often uses **Part-of-Speech (POS) tagging** to identify whether a word is a noun, verb, adjective, etc., ensuring accurate transformations. For instance, "running" as a verb is lemmatized to "run," while "better" as an adjective is lemmatized to "good."

Lemmatization is required to:

- Preserve **linguistic correctness**
- Reduce **vocabulary size**
- Improve **semantic understanding**
- Improve performance in:
    - Information Retrieval
    - Machine Translation
    - Question Answering
    - Text Summarization
    - Sentiment Analysis

Example:

- running → **run**
- better → **good**
- studies → **study**

Original Word	**Lemmatization**

Running	Run
Running	Run
Better	good
Studies	Study
Studies	study

Difference between Stemming and Lemmatization:

Lemmatization	**Stemming**
Converts words to their base or dictionary form (lemma).	Reduces words to their root form (stem), which may not be a valid word.
Higher complexity, context-aware.	Lower complexity, context-agnostic.
Uses dictionaries and morphological analysis.	Uses rule-based algorithms like Porter, Snowball, and Lancaster Stemmers.
Produces more accurate and meaningful words.	Less accurate, may produce non-meaningful stems.
Slower due to more complex processing.	Faster due to simpler rules.
Better search results through understanding context.	Useful for quick search indexing
Essential for tasks needing accurate word forms (e.g., sentiment analysis, topic modeling)	Used for initial stages of preprocessing to reduce word variability
Helps in producing grammatically correct translations	Less common due to potential inaccuracy
Suitable for detailed and precise analysis	Useful for reducing data dimensionality
Example: "Running" → "run", "Better" → "good".	Example: "Running" → "run" or "runn", "Better" → "bett"

Spelling Error Detection and Correction:

Spelling Error Detection:

Spelling error detection in NLP identifies misspelled or invalid words in text to enable subsequent correction. This process is essential for improving the quality of input data in applications like search engines, chatbots, and machine translation.

Words undergo tokenization, then validation against a dictionary or word list; absent words flag as errors. N-gram analysis checks character sequences against frequency matrices to spot anomalies without full dictionary reliance.

Spelling error detection is important because:

- Misspellings reduce **accuracy of NLP systems**
- Affects **search engines and information retrieval**

- Causes errors in:
    - Machine Translation
    - Chatbots
    - Speech Recognition
    - Text Classification
- Essential for **word processors and autocorrect systems**

Types of Spelling Errors:

- **Non-Word Errors:** Non-word errors are spelling errors in which the misspelled word does not exist in the dictionary of a language.
  These errors usually occur due to typing mistakes, pronunciation errors, or lack of spelling knowledge.

  Examples

Correct Word	Misspelled (Non-word)
receive	recive
language	languaeg
school	shcool
computer	compuuter
beautiful	beautifull

- **Real-Word Errors**: **Real-word errors** are spelling errors in which the misspelled word **is a valid word in the dictionary**, but it is **incorrect in the given context**.
  These errors are **hard to detect** because dictionary lookup alone cannot identify them.
  Example:

Correct Sentence	Sentence with Real-Word Error
I will **meet** you tomorrow	I will **meat** you tomorrow
She has **two** books	She has **too** books

Causes of Spelling Errors

- Typographical mistakes
- Phonetic similarity
- Keyboard proximity
- Lack of language proficiency
- OCR errors

Example of Spelling Error Detection

Input Text:

I recieved the mesage yesterday.

Detected Errors:

- recieved
- mesage

Python Code Example (Dictionary-Based Detection)

```
from nltk.corpus import words
import nltk
nltk.download('words')
dictionary = set(words.words())
text = "I recieved the mesage yesterday".split()
errors = []
for word in text:
    if word.lower() not in dictionary:
        errors.append(word)
print("Spelling Errors:", errors)
```

Output:

Spelling Errors: ['recieved', 'mesage']

Spelling Error Correction:

Spelling error correction in NLP detects and fixes misspelled words to improve text processing accuracy. This task enhances downstream applications like search, translation, and sentiment analysis by standardizing input data.
Correction involves two main steps: detecting errors via dictionary lookup or n-gram analysis, then suggesting fixes using similarity measures. Common errors include typos from substitutions, deletions, insertions, or transpositions.

Spelling Error Correction in Natural Language Processing (NLP) is the process of automatically suggesting or replacing incorrect spellings with correct words.
It is performed after spelling error detection.

Spelling Error Correction Methods

1. Dictionary-Based Correction

- Generates candidate words from a dictionary
- Chooses words similar to the misspelled word

Example:

- *mesage → message*

Advantages:

- Simple
- Fast

Limitations:

- Cannot correct real-word errors
- Depends on dictionary size

2 .Edit Distance-Based Correction

Uses **Minimum Edit Distance** (Levenshtein distance) to find the closest correct word.

Edit operations:

- Insertion
- Deletion
- Substitution
- Transposition

Example:

- *speling* → *spelling* (1 insertion)

3. Phonetic-Based Correction

Corrects words based on **pronunciation similarity**.

Algorithms:

- Soundex
- Metaphone
- Double Metaphone

Example:

- *nite* → *night*

Advantage:

- Handles phonetic errors

4. Rule-Based Correction

Uses **hand-crafted linguistic rules**.

Example rules:

- *ie* → *ei* after *c*
- Double consonants

Example:

- *recieve → receive*

5. Statistical Language Model-Based Correction

Uses **n-gram language models** to choose the best word based on context.

Example:

- *there house* ❌
- *their house* ✅

Uses probabilities from:

- Unigram
- Bigram
- Trigram models

6. Machine Learning-Based Correction

Treats spelling correction as a **classification or ranking problem**.

Uses features like:

- Edit distance
- Word frequency
- Context words

Algorithms used:

- Naive Bayes
- SVM
- CRF

7. Neural Network-Based Correction

Uses **deep learning models**:

- RNN
- LSTM
- Transformer models

These models learn both:

- Character-level patterns
- Contextual information

Used in **modern autocorrect systems**.

Minimum Edit Distance:

Minimum Edit Distance , also known as Levenshtein Distance , is a measure of how dissimilar two strings (for example, words or sentences) are by counting the minimum number of operations required to transform one string into the other. The operations typically considered are:

1**. Insertion :** Adding a character to the string.

2. **Deletion** : Removing a character from the string.

3. **Substitution :** Replacing one character in the string with another character.

This concept is widely used in various NLP applications, including spelling correction, DNA sequence analysis, and natural language understanding.

Importance of Minimum Edit Distance

Spelling Correction : Helps in suggesting corrections for misspelled words by calculating how closely a candidate word matches the intended word.

Plagiarism Detection : Measures similarity between texts to identify possible plagiarism or content duplication.

Natural Language Processing Tasks : Useful in tasks such as machine translation and information retrieval.

Calculating Minimum Edit Distance

The minimum edit distance can be calculated using a dynamic programming approach, whichinvolves creating a matrix to track the costs of transforming one string into another.

Steps to Calculate Minimum Edit Distance :

1**. Initialize a Matrix :** Create a matrix where the rows represent the characters of the first string and the columns represent the characters of the second string.

2. **Fill in the Matrix :** The first row and the first column are initialized based on the costs of converting an empty string to a non-empty string (and vice versa). For each cell in the matrix, compute the minimum cost considering the three possible operations (insertion, deletion, substitution).

3. **Traceback :** The value in the bottom-right cell of the matrix will give the minimum edit distance.

Applications of Minimum Edit Distance (Key Points)

- Used for **spelling error detection and correction**
- Helps in **autocorrect and word suggestion systems**
- Improves **search engine query matching**
- Used in **speech recognition** to measure word accuracy
- Corrects errors in **Optical Character Recognition (OCR)**
- Applied in **string similarity and text matching** tasks

- Helps in **morphological analysis** (word variants)
- Used in **plagiarism detection**
- Applied in **DNA/protein sequence alignment**
- Used to evaluate NLP systems (e.g., **Word Error Rate**)

Example:

Error	Candidate Correction	Correct Letter	Error Letter	Type
acress	actress	t	–	deletion
acress	cress	–	a	insertion
acress	caress	ca	ac	transposition
acress	access	c	r	substitution
acress	across	o	e	substitution
acress	acres	–	s	insertion
acress	acres	–	s	insertion

Statistical models:

A statistical model in Natural Language Processing (NLP) uses probability theory and statistics to analyze and predict language patterns from large text data (corpus). Instead of hard rules, it learns from data frequency and probability.

Why Statistical Models are Needed

- Natural language is **ambiguous**
- Rule-based systems are **hard to scale**
- Statistical models can **handle uncertainty**
- They work well with **large corpora**

Types of Statistical models:

1.N-gram Language models

Types

- Unigram (n = 1)
- Bigram (n = 2)
- Trigram (n = 3)

2.Smoothing n-gram models
3. Model evaluation
4. N-gram word models

N-gram Language models:

An **N-gram language model** is a **statistical language model** that predicts the next word based on the previous **(n − 1)** words using probability.

An N-gram language model estimates the probability of a word sequence by considering only the last **n − 1** words.

- **Unigram Model:**
  A **Unigram model** is a **statistical language model** where the probability of a word depends **only on the word itself**, not on any previous words.
  Key Points

- It is a **1-gram (n = 1) language model**
- Assumes **word independence**
- Ignores word order and context
- Probability is based on **word frequency**
- Simplest type of N-gram model

Formula

$$P(w) = \frac{\text{Count}(w)}{\text{Total number of words}}$$

Example
Corpus:
NLP is fun NLP is useful

Word	Count	Probability
NLP	2	2/6
is	2	2/6
fun	1	1/6
useful	1	1/6

Applications

- Baseline language models
- Word frequency analysis
- Simple text classification

Python Example:

```
from collections import Counter

# Sample corpus

corpus = "NLP is fun NLP is useful".split()

# Count word frequencies

word_counts = Counter(corpus)

# Total number of words
```

```
total_words = sum(word_counts.values())

# Calculate unigram probabilities

print("Word\tProbability")

for word, count in word_counts.items():

    probability = count / total_words

    print(f"{word}\t{probability:.2f}")
```

Output:

Word Probability

NLP 0.33

is 0.33

fun 0.17

useful 0.17

- **Bigram Model:**
  A **bigram model** is a type of **statistical language model** in Natural Language Processing (NLP) that predicts a word based on the **immediately preceding word**. It belongs to the family of **n-gram models**, where *n = 2*. Bigram models capture limited contextual information and are widely used in basic NLP applications.
  The bigram model assumes that the probability of a word depends only on the **previous word** (Markov assumption).

$$P(w_1, w_2, \dots, w_n) \approx \prod_{i=2}^{n} P(w_i \mid w_{i-1})$$

The conditional probability is calculated as:

$$P(w_i \mid w_{i-1}) = \frac{\text{Count}(w_{i-1}, w_i)}{\text{Count}(w_{i-1})}$$

Example
Corpus:
NLP is fun NLP is useful
Unigram Counts:

- NLP = 2
- is = 2
- fun = 1
- useful = 1

Bigram Counts:

- (NLP, is) = 2
- (is, fun) = 1
- (is, useful) = 1

Probability Calculation:

$$P(\text{is} \mid \text{NLP}) = \frac{2}{2} = 1$$

$$P(\text{fun} \mid \text{is}) = \frac{1}{2} = 0.5$$

Advantages

- Captures **local word order**
- Simple and computationally efficient
- Improves performance over unigram models
- Easy to implement and understand
-

Limitations

- Data sparsity problem
- Assigns zero probability to unseen bigrams
- Limited context (only one previous word)
- Not suitable for complex language understanding

Applications

- Spell checking
- Speech recognition
- Machine translation
- Text prediction and autocomplete

Python Example:

```
from collections import Counter
corpus = "NLP is fun NLP is useful".split()
bigrams = [(corpus[i], corpus[i+1]) for i in range(len(corpus)-1)]
unigram_counts = Counter(corpus)
bigram_counts = Counter(bigrams)
for bigram, count in bigram_counts.items():
    w1, w2 = bigram
    probability = count / unigram_counts[w1]
    print(f"P({w2} | {w1}) = {probability}")
```

Output:

```
P(is | NLP) = 1.0
P(fun | is) = 0.5
P(useful | is) = 0.5
P(NLP | fun) = 1.0
```

- **Trigram Models:**

  A **trigram model** is a type of **statistical language model** in Natural Language Processing (NLP) that predicts a word based on the **previous two words**. It is an extension of unigram and bigram models and provides better contextual understanding by considering more word history.

  The trigram model is based on the **second-order Markov assumption**, which states that the probability of a word depends only on the **two preceding words**.

$$P(w_1, w_2, \ldots, w_n) \approx \prod_{i=3}^{n} P(w_i \mid w_{i-2}, w_{i-1})$$

The conditional probability is calculated as:

$$P(w_i \mid w_{i-2}, w_{i-1}) = \frac{\text{Count}(w_{i-2}, w_{i-1}, w_i)}{\text{Count}(w_{i-2}, w_{i-1})}$$

Example

Corpus:

NLP is very useful NLP is very easy

Bigram Counts:

- (NLP, is) = 2
- (is, very) = 2
- (very, useful) = 1
- (very, easy) = 1

Trigram Counts:

- (NLP, is, very) = 2
- (is, very, useful) = 1
- (is, very, easy) = 1

Probability Calculation:

$$P(\text{very} \mid \text{NLP, is}) = \frac{2}{2} = 1$$

$$P(\text{useful} \mid \text{is, very}) = \frac{1}{2} = 0.5$$

Advantages

- Captures **more context** than bigram models
- Produces more fluent and meaningful predictions
- Reduces ambiguity in word prediction

Limitations

- Severe data sparsity problem
- Requires large corpus
- High memory and computational cost
- Zero probability for unseen trigrams

Applications

- Language modeling
- Speech recognition
- Machine translation
- Text generation and autocomplete

Python Example:

```
from collections import Counter

corpus = "NLP is very useful NLP is very easy".split()

trigrams = [(corpus[i], corpus[i+1], corpus[i+2]) for i in range(len(corpus)-2)]
```

```
bigrams = [(corpus[i], corpus[i+1]) for i in range(len(corpus)-1)]
bigram_counts = Counter(bigrams)
trigram_counts = Counter(trigrams)
for trigram, count in trigram_counts.items():
    w1, w2, w3 = trigram
    probability = count / bigram_counts[(w1, w2)]
    print(f"P({w3} | {w1}, {w2}) = {probability}")
```

Output:

P(very | NLP, is) = 1.0

P(useful | is, very) = 0.5

P(NLP | very, useful) = 1.0

P(is | useful, NLP) = 1.0

P(easy | is, very) = 0.5

Indian Language Processing in NLP:

Indian Language Processing (ILP) is a subfield of **Natural Language Processing (NLP)** that focuses on the analysis, understanding, and generation of **Indian languages** such as **Hindi, Tamil, Telugu, Kannada, Malayalam, Bengali, Marathi, Gujarati, Punjabi, Urdu**, etc.

India has **linguistic diversity**, so ILP has unique challenges compared to English NLP.

Key Features of Indian Languages

- **Multilingual nature** – India has **22 scheduled languages** and hundreds of dialects
- **Rich morphology** – Words have many forms (gender, number, tense, case)
- **Free word order** – Sentence structure is flexible
- **Script diversity** – Devanagari, Tamil, Telugu, Kannada, Bengali, Perso-Arabic, etc.
- **Agglutinative languages** – Words formed by combining morphemes (Tamil, Telugu)

Levels in Indian Language Processing:

1. Phonological Level

- Deals with sound patterns of a language
- Focuses on pronunciation, stress, and intonation
- Important for speech recognition and text-to-speech systems
- Indian languages have rich phoneme inventories and aspirated sounds

2. Morphological Level

- Studies word formation and internal structure of words
- Handles root words, prefixes, suffixes, and inflections
- Indian languages are morphologically rich

3. Lexical Level

- Deals with individual words and their meanings
- Includes dictionary lookup, POS tagging, and word normalization
- Helps in identifying word categories like noun, verb, adjective

4. Syntactic Level

- Focuses on sentence structure and grammar rules
- Handles free word order common in Indian languages
- Uses parsing techniques to analyze sentence structure

5. Semantic Level

- Deals with meaning of words and sentences
- Resolves ambiguity and determines correct interpretation
- Important for machine translation and question answering

6. Discourse Level

- Analyzes relationship between sentences
- Handles co-reference resolution and coherence
- Important for document-level understanding

7. Pragmatic Level

- Focuses on contextual and situational meaning
- Considers speaker intention, politeness, and social norms
- Very important in Indian languages due to honorific forms

Major Challenges in Indian Language NLP

- Lack of large annotated datasets
- Spelling variations
- Code-mixing and code-switching
- Complex morphology
- Multiple scripts for same language (e.g., Hindi–Urdu)

Applications of Indian Language Processing

- Machine Translation (Hindi ↔ English, Tamil ↔ Telugu)
- Speech Recognition Systems
- Text-to-Speech (TTS)
- Optical Character Recognition (OCR)
- Chatbots in Indian languages
- Sentiment Analysis
- Information Retrieval
- Spell Checkers & Grammar Checkers

Tools and Resources for Indian Language NLP

- Indic NLP Library
- iNLTK
- AI4Bharat datasets
- ILCI Corpus
- UD Treebanks for Indian languages
- BERT-based models (IndicBERT)